COS20007

Object Oriented Programming

Learning Summary Report

Thanh Tam Vo
103487596

Self-Assessment Details

The following checklists provide an overview of my self-assessment for this unit.

	Pass (D)	Credit (C)	Distinction (B)	High Distinction (A)
Self-Assessment				X

Self-Assessment Statement

	Included
Learning Summary Report	X
Test is Complete in Doubtfire	X
C# programs that demonstrate coverage of core concepts	X
Explanation of OO principles	X
All Pass Tasks are Complete on Doubtfire	X

Minimum Pass Checklist

	Included
All Credit Tasks are Complete on Doubtfire	X

Minimum Credit Checklist (in addition to Pass Checklist)

	Included
Distinction tasks (other than Custom Program) are Complete	X
Custom program meets Distinction criteria & Interview booked	X
Design report has UML diagrams and screenshots of program	X

Minimum Distinction Checklist (in addition to Credit Checklist)

	Included
HD Project included	X
Custom project meets HD requirements	X

Minimum High Distinction Checklist (in addition to Distinction Checklist)

Declaration

I declare that this portfolio is my individual work. I have not copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part of this submission been written for me by another person.

Signature: **Thanh Tam Vo**

Portfolio Overview

This portfolio includes work that demonstrates that I have achieve all Unit Learning Outcomes for COS20007-Object Oriented Programming to a **High Distinction** level.

At the end of this semester, I can proudly say about my achievement such as:

- Understanding about the Object Oriented Programming principle (its four concepts will be explained again in the reflection part of this report)
- Using C# and .NET to solve weekly assignments, therefore during this course, I have been building a strong foundation of the .NET framework. It would be beneficial for myself to be a .NET developer in the future.
- Understanding the position of a program tester because we have a lecture called Unit Testing
- Designing UML diagram following Object Oriented design pattern for my final High Distinction project
- Conducting a research related to Object Oriented Programming giving myself an inspiration in pursuing tertiary education

I strongly believe that my effort for this course deserves the High Distinction band score not only because I have a fundamental knowledge of Object Oriented Programming through lectures, but also because the course raises a question and it drives me constantly: *"How to apply Object Oriented thinking into daily life?".* Therefore, I decided to conduct research named *"Object Oriented Programming From Daily Life Perspective"* to clarify this problem(this paper is a submission of task 9.2HD). This research is meaningful for me because it is my first work, also it inspires me to follow an academic career, which is an idea that I have never thought of. Additionally, my final project for the task 6.4HD implements the Object Oriented Programming concepts (the submission of the task 6.4HD elaborates on the implementation), it is a live map of the International Space Station(ISS) from NASA and allows users to record the current position of the ISS through a text file(.txt). After developing the project 6.4HD, I also collected the data related to ISS position on July 31st, 2022 in order to dedicate my work to other scientific research. Here is the link to my collected data.

Now, let's summarize the weekly assignment.

Week 1

- We were introduced to C# programming language for the first time and the task 1.3P is a great assignment to summarize the basic idea of C# by a reference sheet.
- We have to construct an object to print "Hello world" to the terminal.

Week 2

- Creating a Counter class to prepare for the next Clock program
- Using SplashKit to draw the first Rectangle shape
- First iteration applying the Unit Testing knowledge

Week 3

- Demonstrating the memory allocation: Stack and Heap (this topic will be explained in the reflection part because it makes me interested)

- Using SplashKit to draw shape, but this time the drawing class will be in charge of drawing shape to the screen. In week 2, the shape class will be responsible for this task
- Constructing a clock class based on the created Counter class

Week 4

- Drawing multiple shapes: rectangle, circle, and line. This task refer to the implementation of abstraction
- Iteration 2: List of object were introduced for the first time

Week 5

- Iteration 3: adding the bag object
- Iteration 4: creating command class for the user interaction

Week 6

- Iteration 5: using interface in Object Oriented Programming
- Iteration 6(Credit Task): creating location for the game, drawing UML diagram and UML sequence diagram
- Planning and implementing custom project (Distinction Task): my project is a console application tracking the current position of International Space Station
- Planning and implementing custom project (High Distinction Task): based on the idea of the Distinction project, I visualize the current position of the aircraft in the earth map. The details about this project are submitted with the task 6.5HD and 6.6HD.

Week 7

- Summarizing Object-Oriented Programming concept through the concept map
- Iteration 7(Credit Task): creating the path class so that the player can move
- Iteration 8(Credit Task): creating CommandProcessor class as a abstract class for the application of inheritance and polymorphism

Week 8

Semester test

Week 9

Planning and conducting OOP research (High Distinction Task): I decided to conduct a research named "*Object Oriented Programming From Daily Life Perspective*". The details about this paper are submitted with the task 9.1HD and 9.2HD.

Week 10

There are not any tasks that need to finish in week 10

Week 11

- Constructing the clock class as week 3, however this time the programming language should be different, I decided to use Python for completing this task
- Learning Summary Report

Reflection

The most important things I learnt:

After the second semester at Swinburne Vietnam, I have a deep insight into the Object-Oriented concept, which is one of the most important principles that I must learn during my career pathway being a software engineer. In this course COS20007 provided by Swinburne University of Technology, I learned about four pillars of Object Oriented Programming: Abstraction, Encapsulation, Inheritance, and Polymorphism. Moreover, the question *"What is a good Object-Oriented design?"* is mostly raised by the lecturer until the end of the semester. To be honest, it burns my curiosity - how to design a clear model and successfully demonstrate the relationship between objects in the software. Thanks to the weekly assignment and the last game project in the course *COS10009 Introduction to Programming*, there are not any obstacles during the Object Collaboration part.

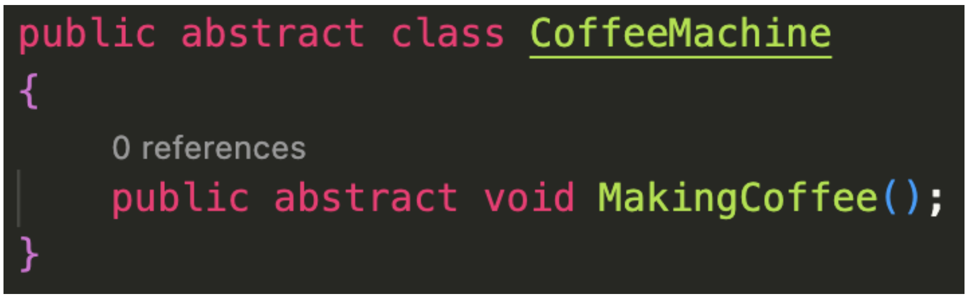
Unlikely the last COS10009 course, in this COS20007 course, we use an object-oriented language like C# instead of Ruby, therefore, the importance of objects and their interactions is more and more emphasized. Also, we were introduced to .NET, which is an open-source developer platform. This is truly beneficial since .NET is still used nowadays and it is not outdated at all. Now I will explain the most important topic of Object-Oriented Programming: its four fundamental concepts: Abstraction, Encapsulation, Inheritance, and Polymorphism.

These four concepts are explained in the Pass task 7.1, however, I would like to break them down into pieces again.

Abstraction

Abstraction is a terminology for hiding information from users, just showing them necessary information without the implementation.

For example: The figure 1 below represent a class called *CoffeeMachine* has a function MakingCoffee, however we do not see any lines of code of that function, all we need to know is that function will create a cup of coffee.



```
public abstract class CoffeeMachine
{
    0 references
    public abstract void MakingCoffee();
}
```

Figure 1. CoffeeMachine class

Encapsulation

This is a process of manipulating data accessibility of the class. The figure 2 below will demonstrate the difference between access specifiers in C# programming language.

Access Specifier	**Characteristic**
public	Its attributes can be accessed by other classes from the same namespace.
private	Its attribute can be only accessed through the property *NOTE: derived class cannot even access
protected	This specifier solves the problem in the 'private' specifier above, a derived class can access the attribute

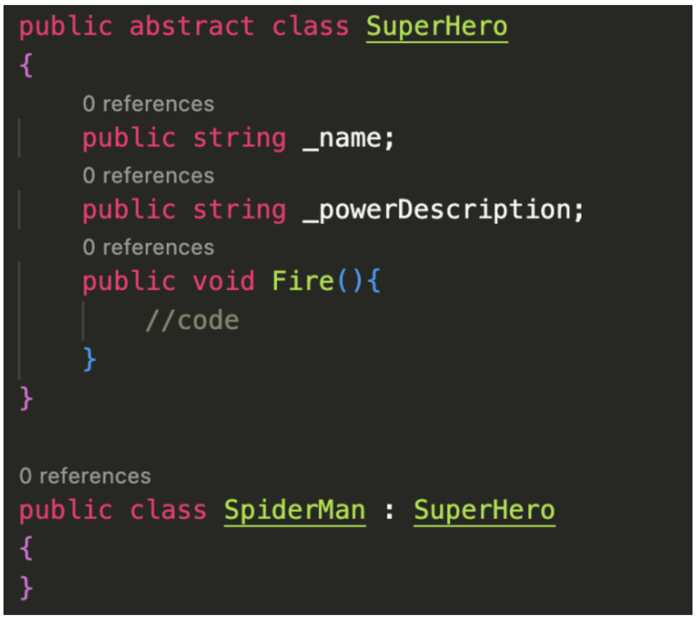
Figure 2. Access Specifier

We can say that Encapsulation is an implementation of abstraction.

Inheritance

As human beings, in Object Oriented Programming, we also have the Inheritance concept, this will facilitate the reusability of code because the derived class will inherit all features from the base class. In other words, we can create a new class based on an existing class.

For example: Figure 3 below will demonstrate the implementation of inheritance in the coding section, the *SuperHero* class is the base class in this situation. Following that, we create a derived class called *SpiderMan* based on the *SuperHero* class, so the *SpiderMan* class can use the function *Fire*



```
public abstract class SuperHero
{
    0 references
    public string _name;
    0 references
    public string _powerDescription;
    0 references
    public void Fire(){
        //code
    }
}

0 references
public class SpiderMan : SuperHero
{
}
```

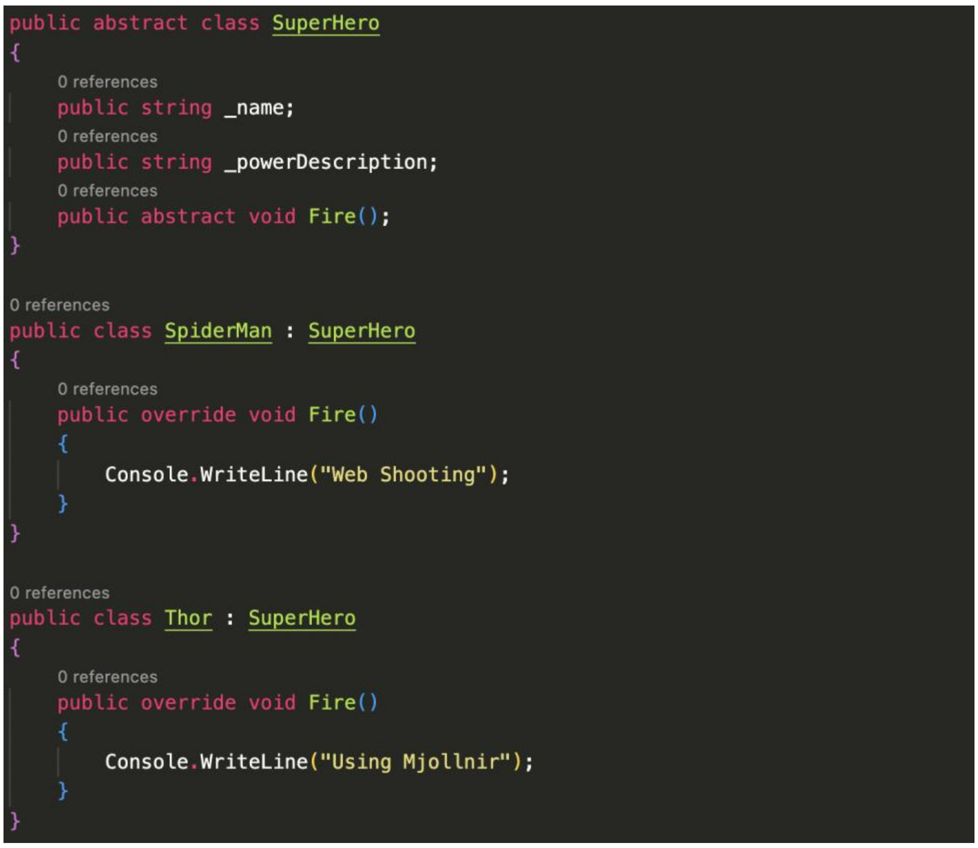
Figure 3. Implementation of Inheritance

There are 4 types of inheritance, which are included in my research project 9.2HD, they are: single inheritance, multilevel inheritance, Hierarchical inheritance, multiple inheritance.

Polymorphism

A base can have multiple derived classes, so these derived classes can have the same method (all of them inherit from the same base class). However, derived classes can perform the method/function differently. We call this phenomenon Polymorphism.

For example: classes called *SpiderMan* and Thor *inherit* from the same class *SuperHero*, however, they perform the method *Fire()* differently. Figure 4 below will demonstrate the idea.



```
public abstract class SuperHero
{
    0 references
    public string _name;
    0 references
    public string _powerDescription;
    0 references
    public abstract void Fire();
}

0 references
public class SpiderMan : SuperHero
{
    0 references
    public override void Fire()
    {
        Console.WriteLine("Web Shooting");
    }
}

0 references
public class Thor : SuperHero
{
    0 references
    public override void Fire()
    {
        Console.WriteLine("Using Mjollnir");
    }
}
```

Figure 4. Polymorphism demonstration

The things that helped me most were:

First, I personally appreciate Dr. Dustin for delivering this course COS20007 to Swinburne Vietnam. Thanks to him, I can understand the Object Oriented Programming concept and complete my final project with his advice. He successfully explained what is a good Object Oriented design pattern, which is a topic I will expand on in the next section, and applied it to coding to guarantee that we can understand.

Same as developers, Stack Overflow is a great platform for me. My problem can be solved thanks to the platform. Moreover, I can connect to more than one million developers around the world in order to expand my networking in my future career.

I found the following topics particularly challenging:

The learning outcome of this course is achieving good object-oriented design pattern thinking. The question "*What is a good Object Oriented design?*" is repeated over the course, plus, Swinburne University of Technology hosted a unit chair with Dr. Charlotte Pierce to dive into this topic. To me, the design pattern is the most challenging. As a future developer, I have to design a solution (or model) for many problems that exist in different OOP contexts.

Although having a clear explanation from Dr. Dustin and a Unit Chair with Dr. Charlotte Pierce, I still find it a bit difficult in this field, however, I will shortly summarize what is a good object-oriented design and what factors play a significant impact on our design.

Three rules in Object Oriented design should be considered:

- *A class should not be responsible for multiple jobs*
  For example, a class call Card should not be in charge of shuffling
- *A class/object should be introverted*
  For example, an object called Card does not pay attention to other Card objects, in other words, there will not be any interaction between objects from the same class.
- *Derived classes should follow the conformity*

Achieving low coupling and high cohesion
The target of designing a software model is "*low coupling and high cohesion*", as the three rules above, the terminology "*low coupling*" means that a component should not depend on others, and a modification of a class should not affect other classes.

On the other hand, "*high cohesion*" means that the responsibility of each class should be clear and distinct, two classes must not be in charge of the same job.

To conclude, I admit that in order to achieve a good OOP design pattern mindset, it requires a lot of effort and practice.

I found the following topics particularly interesting:

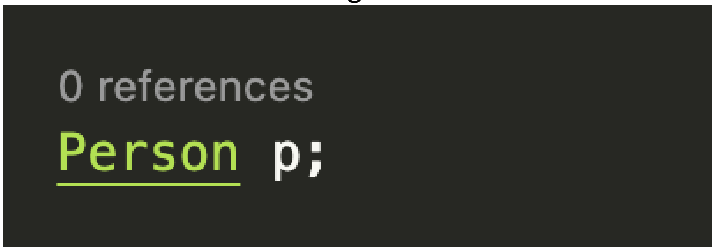
Creating a UML diagram is the first step in solving an OOP problem, a clear diagram can lead to a perfect solution. This section makes me interested since it requires not only coding skills but also problem-solving skills. Additionally, during the UML designing section, I can practice the three rules principle in OOP design above and improve myself as a software engineer.

Moreover, I have learned about the dotnet Heap and Stack through the Canvas Object Oriented Material. This knowledge is really helpful for my understanding about computer systems. Long story short, the computer memory can be separated into two areas: Heap and Stack.

There are two type of variable in C#, which is the main programming language that I use in this course:

- local variable/reference variable: being allocated at the Stack
- Instance class: being allocated at the Heap

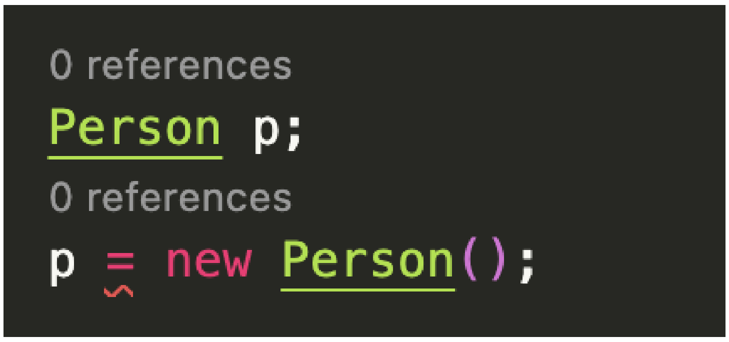
For example, when we declare a variable as figure 5 below



```
0 references
Person p;
```

Figure 5. Declaring a local variable

The variable *p* will be stored at the stack, then we will construct the object Person as figure 5.1 below



```
0 references
Person p;
0 references
p = new Person();
```

Figure 5.1 Constructing the Person object

Now, let’s see what happens in the memory. The figure 5.2 will depict our situation

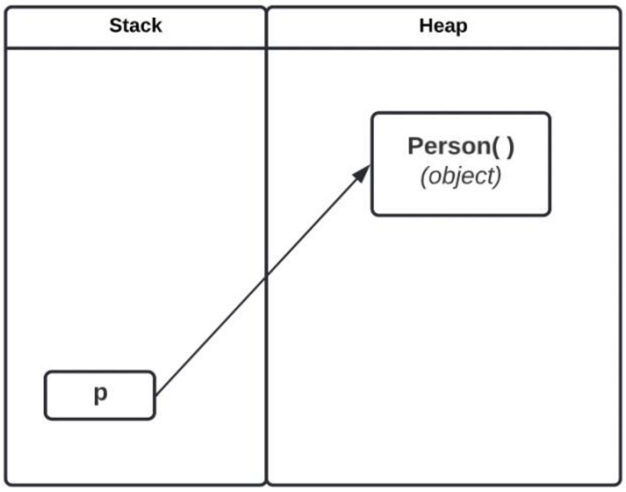


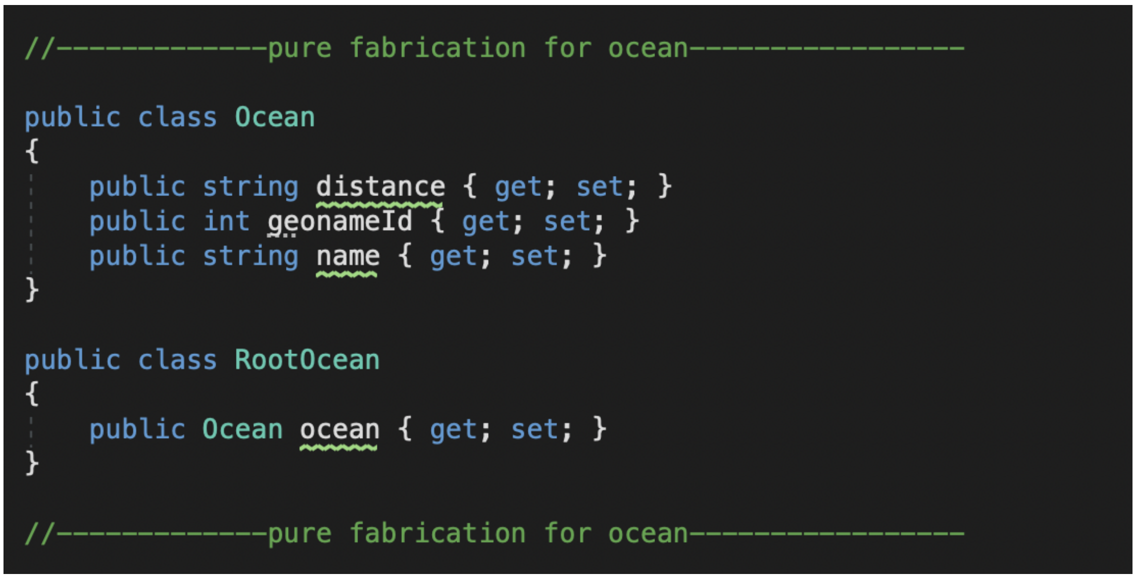
Figure 5.2 Variable allocation

Now, p is considered as a reference variable

I feel I learnt these topics, concepts, and/or tools really well:

In week 9 of the semester, I was introduced to the Pure Fabrication concept. This topic plays an important role in my final project. In general, the explanation for Pure Fabrication is creating a temporary class to store information without any relation to our main components.

The following figure 6 will show the application of Pure Fabrication in my project



```
//-------------pure fabrication for ocean-----------------

public class Ocean
{
    public string distance { get; set; }
    public int geonameId { get; set; }
    public string name { get; set; }
}

public class RootOcean
{
    public Ocean ocean { get; set; }
}

//-------------pure fabrication for ocean-----------------
```

Figure 6. Application of Pure Fabrication concept

The Ocean class above will store all the information needed from the internet, indeed, it is a decoding JSON object, my project application just needs the attribute *name* from the class Ocean.

I still need to work on the following areas:

- Object Oriented design pattern: as I mentioned before, this topic is the most challenging. Swinburne Canvas provides many patterns but I have not yet applied them to coding sessions. There are many design patterns that I need to work on: adapter patterns, creational patterns, factory patterns, etc.
- Using Unity Engine for game development: being a future software engineer, this course is a perfect occasion to learn about one of the most popular engines in Game Design: Unity. However, I was too busy optimizing my UML diagram for the final project.

My progress in this unit:

Unfortunately, all the tasks were submitted through Canvas but not Doubtfire, therefore we do not have any graph reflecting my progress. However, during the course, all the Pass, Credit, Distinction, High Distinction tasks were submitted without any late marks. Plus, I did not apply any extension to my work during the course. This circumstance proves that I can handle the tasks and organize working time precisely.

This unit will help me in the future:

- Achieving strong foundation of Object Oriented Programming concepts, which is one of the most important topics in software engineering.
- Applying C# and dotnet to personal project in order to increase the variety of my programming portfolio
- Time managing skill, priority organizing: Pass tasks are always the priority.

If I did this unit again I would do the following things differently:

I would say that if I had a chance to enroll in this course again, I would spend my time developing games using the Unity engine. As I said before, I still need to work on Game Designing. There are not any reasons that make me refuse to learn one of the most popular game engines in the world.